EPL646 – Advanced Topics in Databases
Spark

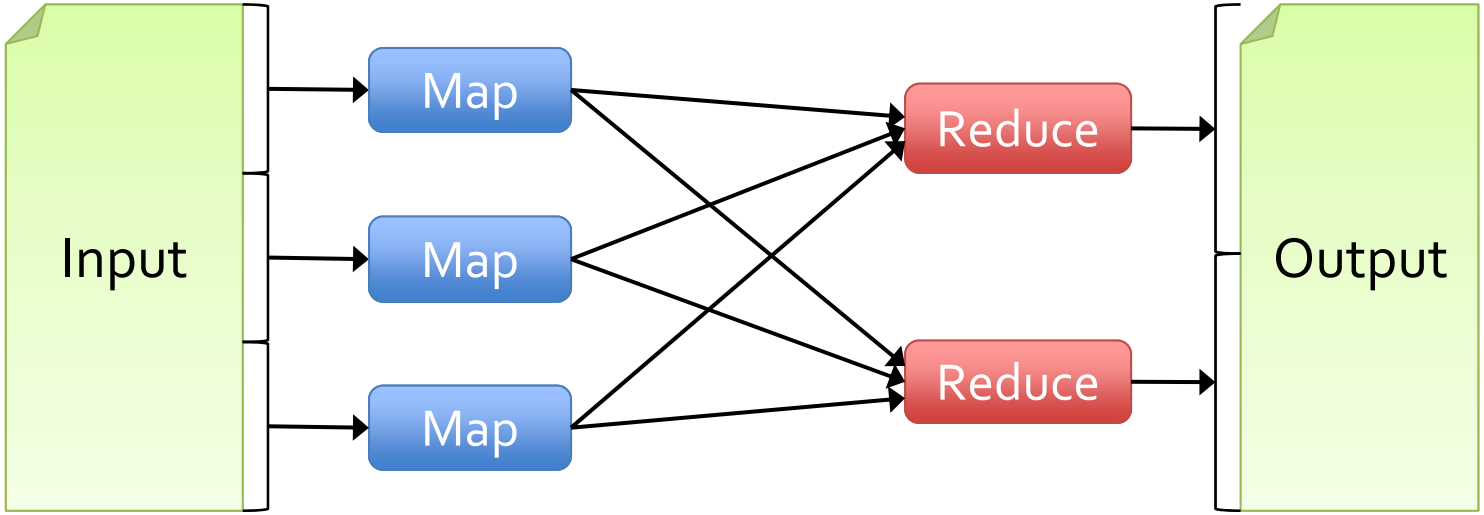
http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html

Introduction to Apache Spark

- Another cluster computing framework
- Developed in the AMPLab at UC Berkeley
- Started in 2009
- Open-sourced in 2010 under a BSD license
- Proven scalability to over 8000 nodes in production

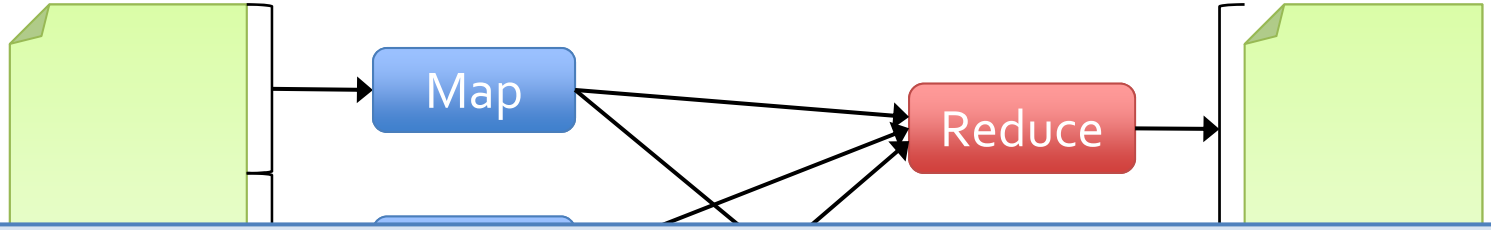
Motivation

- Current popular programming models for clusters transform data flowing from stable storage to stable storage
- E.g., MapReduce:



Motivation

- Current popular programming models for clusters transform data flowing from stable storage to stable storage
- E.g., MapReduce:



Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Key points about Spark

- Native integration with Java, Scala and Python through its API
- Acyclic data flow is a powerful abstraction, but is not efficient for applications that repeatedly reuse a *working set* of data:
  - **Iterative** algorithms (many in machine learning)
  - **Interactive** data mining tools (R, Excel, Python)
- Spark makes working sets a first-class concept to efficiently support these apps
- Map/reduce is just one set of supported constructs (structured and relational query processing, ML)

Spark Goal

- Provide distributed memory abstractions for clusters to support apps with working sets
- Retain the attractive properties of MapReduce:
  - Fault tolerance (for crashes & stragglers)
  - Data locality
  - Scalability

Solution: augment data flow model with "resilient distributed datasets" (RDDs)

Generality of RDDs

- We conjecture that Spark's combination of data flow with RDDs unifies many proposed cluster programming models
  - *General data flow models:* MapReduce, Dryad, SQL
  - *Specialized models for stateful apps:* Pregel (BSP), HaLoop (iterative MR), Continuous Bulk Processing
- Instead of specialized APIs for one type of app, give user first-class control of distributed datasets

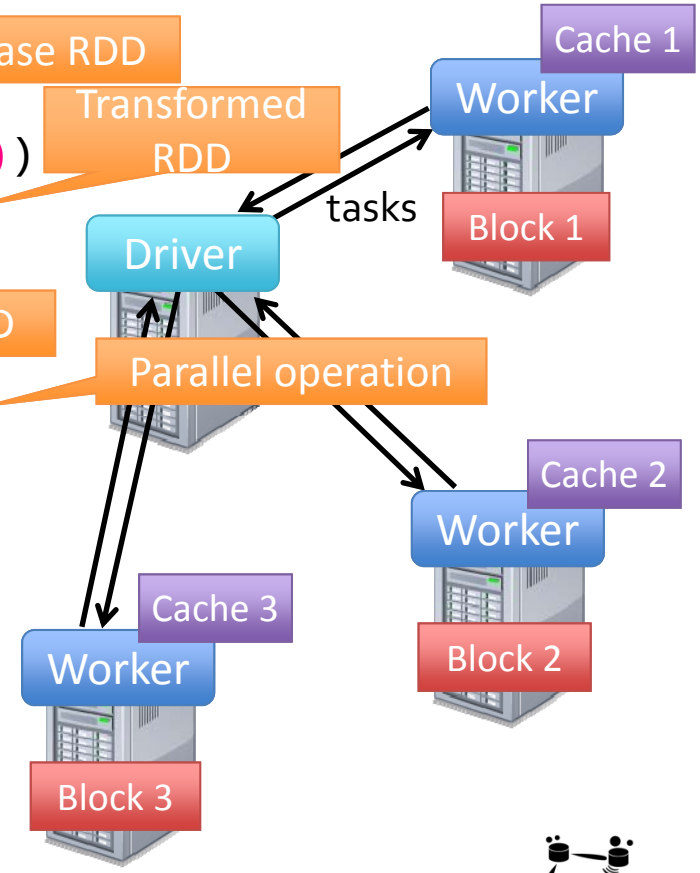
Programming Model

- Resilient distributed datasets (RDDs)
  - Immutable collections partitioned across cluster that can be rebuilt if a partition is lost
  - Created by transforming data in stable storage using data flow operators (map, filter, group-by, …)
  - Can be *cached* across parallel operations
- Parallel operations on RDDs
  - Reduce, collect, count, save, …
- Restricted shared variables
  - Accumulators, broadcast variables

Example: Log Mining

- Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(_.startsWith(“ERROR”))
messages = errors.map(_.split(‘\t’)(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains(“foo”)).count
cachedMsgs.filter(_.contains(“bar”)).count
. . .
```



Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

RDDs in More Detail

- An RDD is an immutable, partitioned, logical collection of records
  - Need not be materialized, but rather contains information to rebuild a dataset from stable storage
- Partitioning can be based on a key in each record (using hash or range partitioning)
- Built using bulk transformations on other RDDs
- Can be cached for future reuse

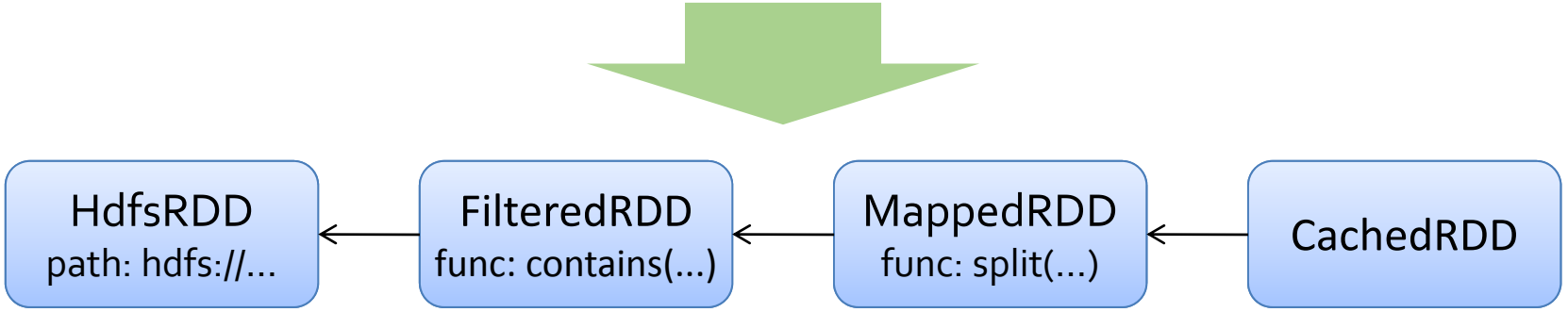
RDD Operations

Transformations (define a new RDD)	**Parallel operations** (return a result to driver)
map filter sample union groupByKey reduceByKey join cache …	reduce collect count save lookupKey …

RDD Fault Tolerance

- RDDs maintain lineage information that can be used to reconstruct lost partitions
- Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```

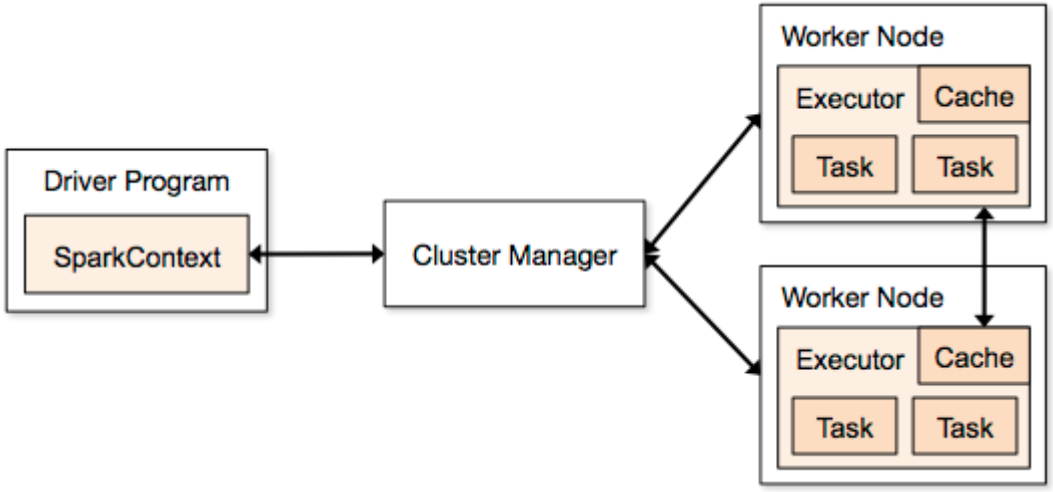


Benefits of RDD Model

- Consistency is easy due to immutability
- Inexpensive fault tolerance (log lineage rather than replicating/checkpointing data)
- Locality-aware scheduling of tasks on partitions
- Despite being restricted, model seems applicable to a broad variety of applications

Components

1. connects to a cluster manager which allocates resources across applications
2. acquires executors on cluster nodes – worker processes to run computations and store data
3. sends app code to the executors
4. sends tasks for the executors to run



Hands on - Spark Shell

1. start-all.sh
2. cd /usr/local/spark
3. ./bin/spark-shell

From the “scala>” REPL prompt, let’s create some data:

>*val data = 1 to 10000*

Then create an RDD based on that data:

>val distData = sc.**parallelize**(data)

Finally use a filter to select values less than 10:

>distData.filter(_ < 10).collect()

Hands on - Spark Shell

1. Now lets run word count example:

- Select the input from HDFS

```
>val f = sc.textFile("/input/dataset/pg4300.txt") //on hdfs
```

- Create the map function

```
>val words = f.flatMap(l => l.split(" ")).map(word => (word, 1))
```

- Create the reduce function

```
>words.reduceByKey(_ + _).collect.foreach(println)
```

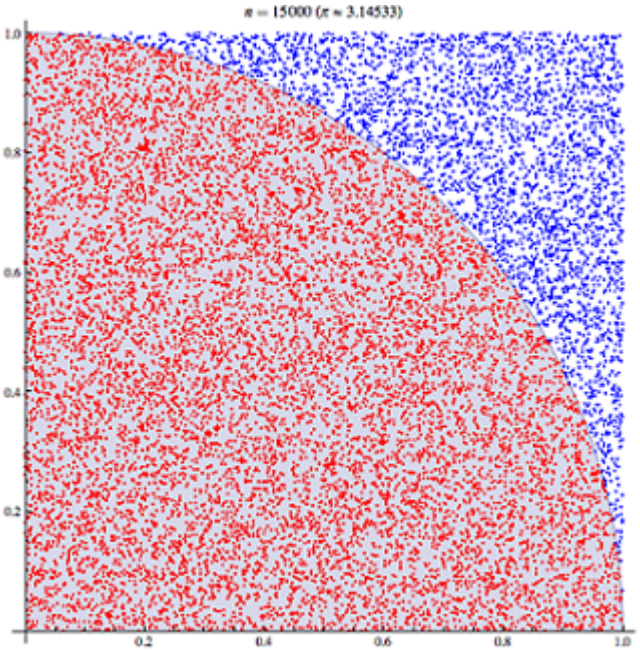
- Save the result on hdfs

```
>words.reduceByKey(_ + _).saveAsTextFile("words_output") //on hdfs
```

Spark Examples: Estimate PI

- Now, try using a Monte Carlo method to estimate the value of Pi

  ./bin/run-example SparkPi 2 local



Spark Examples: Estimate Pi

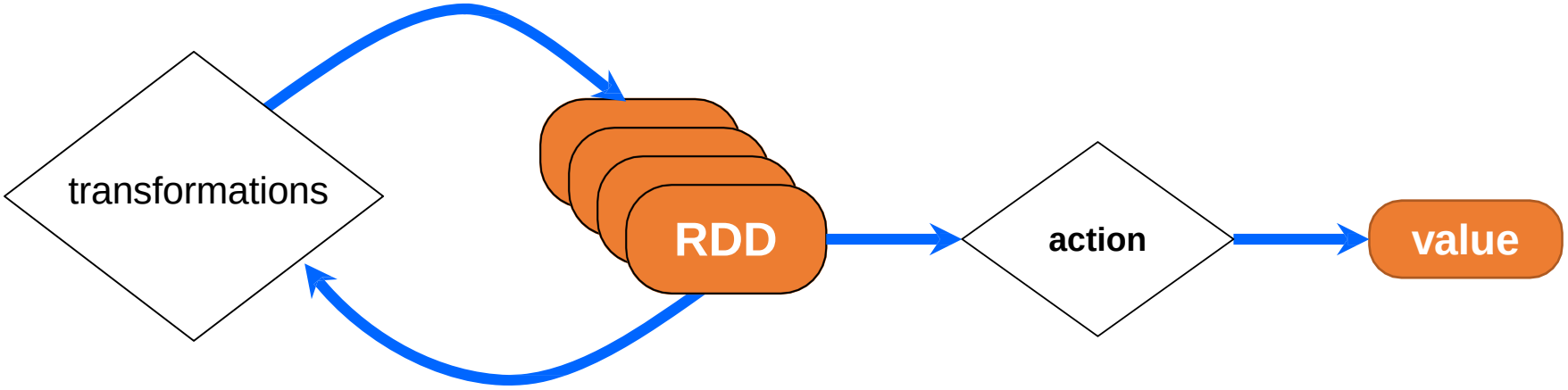
```scala
import scala.math.random
import org.apache.spark._

/** Computes an approximation to pi */
object SparkPi {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("Spark Pi")
    val spark = new SparkContext(conf)
    val slices = if (args.length > 0) args(0).toInt else 2
    val n = 100000 * slices
    val count = spark.parallelize(1 to n, slices).map { i =>
      val x = random * 2 - 1
      val y = random * 2 - 1
      if (x * x + y * y < 1) 1 else 0
    }.reduce(_ + _)
    println("Pi is roughly " + 4.0 * count / n)
    spark.stop()
  }
}
```

Spark Examples: Estimate Pi

```
val count = spark.parallelize(1 to n, slices)
```
Base RDD

```
.map { i =>
  val x = random * 2 - 1
  val y = random * 2 - 1
  if (x * x + y * y < 1) 1 else 0
}
```
transformed RDD

```
.reduce(_ + _)
```
action



Questions?

http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html